# TAS-PLUS

---

# RELATIONAL DATABASE

## Developer's Version

## COPYRIGHT

Copyright by Business Tools, Inc. 1986. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means, without the written permission of Business Tools, Inc.

## TRADEMARKS

TAS, TAS-Plus, The Accounting Solution and Business Tools are trademarks of Business Tools, Inc. IBM, PC, XT, AT are trademarks of International Business Machines Corporation. dBASE II and dBASE III are trademarks of Ashton-Tate. Wordstar and MailMerge are trademarks of Micropro International Corporation. Microrim, R:BASE SERIES, and CLOUT are trademark of Microrim; Inc. MS-DOS is a trademark of Microsoft, Inc.

## DISCLAIMER

This TAS-Plus User Manual is printed in the U.S.A. Any names or references to persons, corporations, or products mentioned in the TAS-Plus User Manual are intentionally fictitious. Any resemblance to persons living or dead or to actual corporations or products is purely coincidental. Business Tools, Inc. believes that the information contained in the manual is correct. However, Business Tools, Inc. does not assume responsibility for the accuracy of the content of the TAS-Plus User Manual nor use of the manual, nor for any patent infringements or other rights of third parties who may use the manual or the software herein. Business Tools, Inc. reserves the right to revise, update or change either the product or the manual without a direct or inferred obligation to Business Tools, Inc. to notify any person of such revisions, updates, or changes.

## SOFTWARE COPYRIGHT NOTICE:

This product is only licensed and not sold. It remains the property of Business Tools, Inc. This package contains a written limited warranty, software license agreement, and a sealed system diskette package. If you cannot agree to the software license, return the product to the point of purchase within three (3) days for a full refund, provided all items are returned in the new condition and provided the system diskette package is still sealed.

Unauthorized use of the software or related materials can result in civil damages and criminal penalties.

# Acknowledgments

Business Tools, Inc. wishes to acknowledge the contributions made by these people towards the development of TAS-Plus.

**PRODUCT DEVELOPMENT**

Philip D. Mickelson

**DOCUMENTATION**

Rick Grafton
Craig G. Ingle
Jennifer S. Rowe

# TAS-Plus

## Relational Database

## Developer's Version

---

## Table of Contents

### REFERENCE GUIDE


I. List of TAS-Plus Developer Commands .....1

II. Accessing Files Using Variable File Numbers .....29

III. Memory Storage Areas .....31

IV. Conversion of TAS-Plus Editor Commands to ASCII Text Editor Type Commands .....33

V. Creating Screen and Report Formats for TAS-Plus using an ASCII Text Editor .....47

VI. Cross Reference Listing of TAS-Plus Command Numbers and Titles .....51

VII. Trace Utility .....57

VIII. Multiuser Operations .....59

# TAS-PLUS DEVELOPER'S VERSION

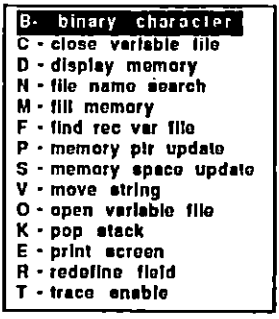
---

# REFERENCE GUIDE

# I. List of TAS-Plus Developer Commands

The following is a list of the commands available in the TAS-Plus language. They are in alphabetical order. The command name is first on the page followed by a picture of the choice you would make in the editor to use this command. Also included are the definitions of the parts of the command and examples.

# binary character




The **binary character** is used to create a single character alphanumeric value that can be displayed on the screen as a graphic character or screen attribute. It may also be compared to a value that was read from a file. The command will allow conversion from a numeric to a single character alphanumeric or back again.

Note: The maximum value that may be converted is 255.

**PARTS OF THE COMMAND:**

**alpha field** - This is the single character alphanumeric field that will receive the binary character or has it already. This should be a 1 character alphanumeric field. This entry is required.

**numeric field** - This is the field that contains the value to be converted to binary or will receive the numeric value of the binary character. This should be a 3 character number field with no decimal characters. This entry is required.

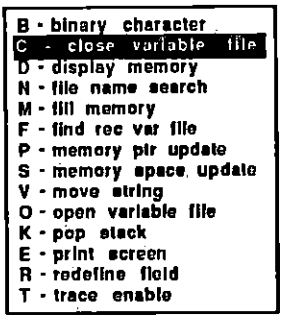
**alpha to numeric or numeric to alpha - A** converts a binary character to its normal numeric equivalent. **N** converts from a normal numeric number to its single character binary equivalent. This is required.

**TAS EDITOR EXAMPLE:**

binary chr: alpha field: ALPHA numeric field: NUMBER
cvt from alpha to numeric

binary chr: alpha field: ALPHA numeric field: NUMBER
cvt from numeric to alpha

# close variable file




The **close variable file** command is used to close a file that was opened previously using the **open variable file** command. This is not necessary since the program will automatically close the file for you when the next program is run. However, if the file being closed is the last file opened then the buffer space used for this file will be reclaimed and the next file opened will use that space. This would allow you to open and close many different files without the fear of running out of memory space.

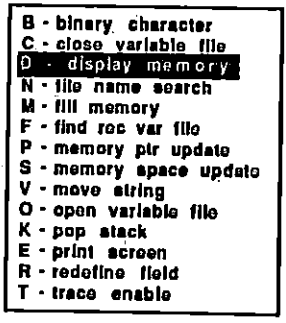
**PARTS OF THE COMMAND:**

**file num** - The number for the file that was received in the **open variable file** command. This entry should be a 2 character numeric field and is required.

**TAS EDITOR EXAMPLE:**

close var file: file num: NUMBER

# display memory




This command is similar to the **display array** command. The difference is this command is used to display information that is being stored in one of the memory storage locations.

Note: This command will do no translation of data. All information displayed on the screen using this command should be in alphanumeric form already. If you try to display a BCD numeric it will appear as a graphics character.

**PARTS OF THE COMMAND:**

**mem loc** - This is the memory storage location number. May be from 1 to 4. Required.

**start loc** - The starting position within the specific memory storage area. The first character within the area is position 1. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**element length** - The length of each 'line' to be displayed. This may be longer than what will fit on the screen. Only that portion that will fit will be displayed. Wrapping will not occur. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**offset** - The position within the element you want to start with. The character at this position will be the first character displayed at the column specified. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**column** - The starting column number. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**row** - The starting row number. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**width** - The number of characters to be printed on a single line. This should be less than or equal to the length of a single element. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

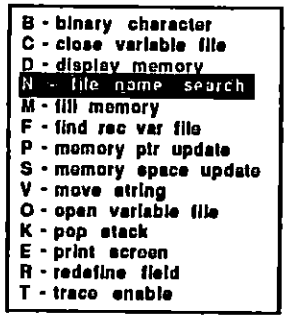
**number of lines** - The number of lines to be printed. Each new line will be the next element in the memory storage area. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**TAS EDITOR EXAMPLE:**

display memory: mem loc: 1 start: START.FLD element len: E.LENGTH offset: CHR.OFFST col: X.POS row: Y.POS width: NUM.CHRS lines: NUM.LNES

display memory: mem loc: 2 start: 1 element len: 30 offset: 1 col: 10 row: 2 width: 30 lines: 10

# file name search




This command is used for finding a file name or a group of file names on a certain drive and directory. You may use the standard PC-DOS wild card characters (? and *). If the file name is not found then the field will be returned blank. This is helpful if you are trying to determine if a file exists. You would put the exact file name into the field and then execute this command. The next line would check to see if the field is empty (""). If it is then you know that the file doesn't exist. If the file name is still there then it does exist. By using this feature and the **menu** command you could create a menu of all programs within a group or on a certain path and drive location.

Note: Before you can use the **next** option you must execute the command with the **first** option specified.

**PARTS OF THE COMMAND:**

**file name** - This field would contain the file name being searched for or the wildcard representation. This is any legal alphanumeric field using the standard format of *fieldname[ss]*. Required.

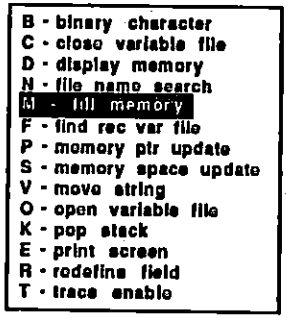
**first / next** - **F** find the first occurrence of the file name. **N** find the next occurrence of the file name that satisfies the wild card specification.

**TAS EDITOR EXAMPLE:**

file name search: file name: WILD.CARD FIRST

file name search: file name: WILD.CARD NEXT

# fill memory




This command will fill a memory storage location, or portion of, with a specific character. This command would generally be used if you wanted to prepare a location for any reason.

Note: The data residing in any memory storage location is **not** predefined. If nothing has been stored in it previously then it will probably contain garbage.

**PARTS OF THE COMMAND:**

**mem loc** - The memory storage location number. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. The value must be 1 thru 4. Required.

**start** - The starting location within the memory storage area. The first position is 1. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**num chrs** - The number of characters to fill. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

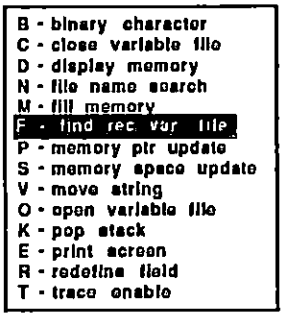
**fill chr** - The character to be used. This is any legal alphanumeric field in the format of *fieldname[ss]* or alphanumeric constant. This should be a 1 character field or constant. Required.

**TAS EDITOR EXAMPLE:**

fill memory: mem loc: LOC.NUM start: START.POS num chrs: NUMB.CHRS fill chr: ALPHA.CHR

fill memory: mem loc: 1 start: 100 num.chrs: 20000 fill chr: "A"

# find rec var file




This command will allow you to find a record in a file that has been opened previously using the **open variable file** command. Fields may be retrieved from the buffer by using the **equals portion of** command, the **redefine field** command or the **move string** command.

**PARTS OF THE COMMAND:**

**file num** - This is the number of the file that was returned in the **open variable file** command. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**key num** - This is the actual key number as specified in the data dictionary. This can be determined by loading the correct record for the key field, using the key field name, and examining the value of the field **DICT.KEYN**. Note: If you want to use the record number you would set the value of this to 0. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**search val** - The value you want to search for. This must be an alphanumeric field in the format of *fieldname[ss]*. Note: If you are searching for a numeric value (actual numeric, date, time or specific record number), you would still make this an alphanumeric field. To set the value

accurately you would use the **put portion of** command to place the correct representation of the field into the search field. Please refer to the chapter in this manual on using the **equals portion of** and **put portion of** commands.

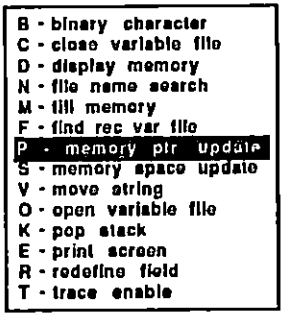
**search type** - **M** exact search. **G** generic search. **B** find beginning record. **E** find ending record. **N** find next record (must do a specific search or B or E first). **P** find previous record (must do a specific search or B or E first).

**error label** - The label of the line to go to if an error occurs while trying to find the record.

**TAS EDITOR EXAMPLE:**

find record var file: file num: FILE.NUMBER key num: KEYN search val: SRCH.FLD search type: M error label: SRCH.ERR

# memory ptr update




This command is used to update a large group of pointers that are being maintained in one of the memory storage locations. This would occur only if you needed to maintain more pointers then would normally fit into a standard TAS-Plus array field (255 elements). You would be able to accomplish the same task by using the **equals portion of** and **put portion of** commands along with the **equals** command however it would be a much more complicated procedure and would take much more time. This command will do up to 5000 calculations in less then 1 second.

This command was created to answer a need in the TAS-Plus source code editor. To be able to make the product as speedy as possible we decided to maintain the entire source code file in memory. To be able to jump to a specific line number we needed to keep a group of pointers that would designate the beginning of the line and another that had the line length. Since some programs can become quite long we could not limit the number of lines to 255 and so decided to use one of the memory segments to hold the pointer list. We found that updating that list as often as we might need to would be too time consuming. (We have an overriding desire for speed here at Business Tools.) With this command and the **memory space update** command you are able to add, delete, and change lines in the source code editor and have it updated instantly. We decided to provide you with these commands and others since we have found that our customers very often find uses for some commands that we

never expected. Also, if you are going to write complex, sophisticated programs you will need these types of command.

Note: The pointers may be separated by other data; however, they must be equidistant from each other.

**PARTS OF THE COMMAND:**

**pointer size** - the actual size of the pointer number. This would be the calculated size not the defined size. For example, a pointer defined to be 8 characters long would actually only be 4 characters when stored. To determine the calculated size (also referred to as csize) of a number you divide the defined size by 2. If there is a remainder add 1 to the result. If the defined size is 5 the csize will be 3. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**start loc** - The starting location within the specific memory segment. The first character position is 1. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**memory loc**- The number of the memory segment. This may be from 1 thru 4. Required.

**amt to add/sub** - The value to add or subtract to/from each pointer. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**occurs every # chrs** - The number of csize chrs from the start of the pointer to the next pointer. For example, if two 5 character defined size pointers made up a group (the target pointer occurred every other time) the answer for this item would be 6. Each pointer would have a csize of 3 for a total csize for the group of 6. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**how many ptrs** - The number of pointers to update. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

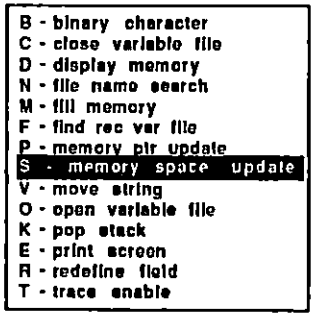
**what to do (AS)** - **A** adds the amount to each of the pointers. **S** subtracts the amount from each of the pointers.

**TAS EDITOR EXAMPLE:**

update memory pointers: ptr size: P.SIZE start loc: P.START mem loc: 2 amt to add/sub: CHANGE occurs # chrs: ELEM.SIZE how many: QTY

update memory pointers: ptr size: 3 start loc: 1 mem loc: 1 amt to add/sub: CHANGE occurs # chrs: 6 how many: TOT.NUM

# memory space update




This command will add space or delete characters from a memory segment. The result of using the command is to move a block of characters either up or down depending on whether the command is to add or subtract.

This command like the **memory ptr update** command previous was created to manage large groups of data in a memory segment area. By using this command you will be able to add information to or subtract information from large data groups very quickly. This command will move any block of data instantly.

**PARTS OF THE COMMAND:**

**memory loc** - the memory segment number. Must be from 1 thru 4.

**number of chrs to add/sub** - The number of characters you want to add at this particular location or the number of characters you want to delete. This is any legal numeric field in the format of fieldname[ss] or numeric constant. Required.

**start loc** - This is the character location you want to start with. The first position is 1. This is any legal numeric field in the format of fieldname[ss] or numeric constant. Required.

**stop loc** - This is the ending location. This is any legal numeric field in the format of fieldname[ss] or numeric constant. Required. Note: The block defined by the starting location and the stop location is going to be moved the number of characters specified in the **number of chrs** field above.

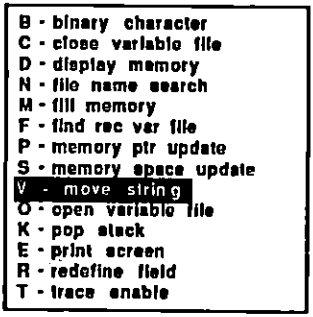
**what to do (AS)** - A adds characters at the start location or moves the block up. S subtracts characters at the start location or moves the block down. Note: In the S mode the block is moved so that the character present after execution at the start loc was at the character position start loc + numb of chrs - 1 before execution.

**TAS EDITOR EXAMPLE:**

add/delete chrs from mem buffer: mem loc: 1 # of chrs: NUM.CHRS start loc: START stop loc: STOP ADD

add/delete chrs from mem buffer: mem loc: 2 # of chrs: 100 start loc: 1 stop loc: 24000 DELETE

# move string




This command is used for moving a large amount (more than one field) from a file buffer to another file buffer, to a memory segment location, or from one memory segment location to another.

**PARTS OF THE COMMAND:**

**frm s/f #** - The from segment/file number. File numbers will always be greater than 4 and segment numbers will always be 1 thru 4. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**frm offset** - The starting location within the file buffer or memory segment location. The first position in either is 1. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**to s/f #** - The to segment/file number. File numbers will always be greater than 4 and segment numbers will always be 1 thru 4. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**to offset** - The starting location within the file buffer or memory segment location. The first position in either is 1. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

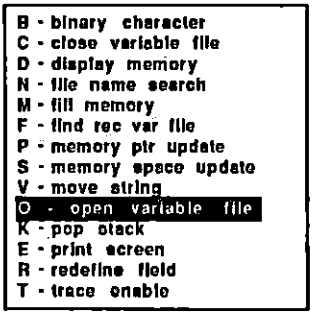
**num of chrs** - The number of characters to move. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**TAS EDITOR EXAMPLE:**

move string: frm s/f #: STRT.FILE frm offset: STRT.LOC to s/f #: TO.FILE to offset: TO.LOC num chrs: HOW.MANY

move string: frm s/f #: 1 frm offset: 1 to s/f #: 2 to offset: 1000 num chrs: 5000

# open variable file




This command will allow you to open a file using a variable name instead of having to know what the file name is when creating the program. This command was instrumental in being able to create the programs **TASADREC** and **TASBRWSE**. There is more overhead when accessing files using this method but it is pretty straight forward and a further explanation is in Chapter II of this manual.

Note: When the file is opened buffer space is provided by the program. If you close this file before you open another the space will be reclaimed. This will also work even if you open another file after this one is opened as long as the last file opened is closed first (last opened / first closed). This, however, only pertains to files opened using this command.

**PARTS OF THE COMMAND:**

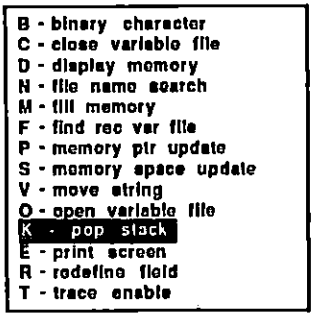
**file name** - The field representing the file to be opened. The value of this field should be just the first 8 characters max of the file name. Just as though you were using the normal **open TAS file** command. This must be any legal alphanumeric field in the format of fieldname[ss].

**file num** - The number representing this file will be returned in this field by the system.

**TAS EDITOR EXAMPLE:**

open var file: field name: VAR.FILE file num: VAR.FNUM

# pop stack




This command has the effect of getting rid of the last gosub. If you were to execute several gosubs and for one reason or another decided not to use the **ret** command to return from the gosub this command will remove the last gosub from the list (also known as a stack). The command, in effect, does a return and then continues executing with the next line after the **pop stack** command instead of the next line after the last gosub command. This command should only be used sparingly, if at all, in most programs. If you should attempt to use this command and a **gosub** was not done previously the program will give the error message that a return was attempted without a previous gosub and the program will exit to the menu. You should also be sure that you are not 'pop'ing the wrong gosub. This command should be used with caution.

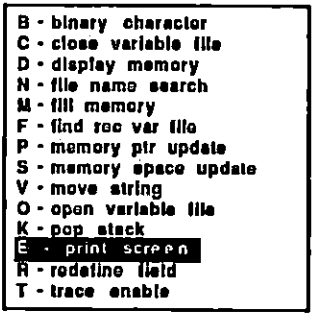
**PARTS OF THE COMMAND:**

**none**

**TAS EDITOR EXAMPLE:**

pop return off stack

# print screen




This command will have the same effect as pressing the PrtSc key on your keyboard.

Note: It will not affect the internal number of lines printed counter.

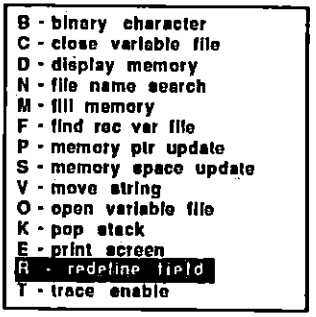
**PARTS OF THE COMMAND:**

none

**TAS EDITOR EXAMPLE:**

print screen

# redefine field




This command will allow you to specify that a field defined earlier in the program using a data dictionary definition or using the **define field** command has been changed. You may change any feature of that field including specifying that it now belongs to a different file or the key number. This command is generally used in conjunction with the **open variable file** command but may also be used on its own.

Note: Don't forget that the field being redefined must be at least as large as the new size unless it is being assigned to a file.

**PARTS OF THE COMMAND:**

**field to be redefined** - The name of the previously defined field. This is any legal alphanumeric field in the format of *fieldname[ss]* or alphanumeric constant. Required.

**field type** - **A** alphanumeric field will accept up to 255 alpha or numeric characters. **N** numeric field will accept up to 20 numeric characters with an even number of decimal characters. If total length is an odd number of characters, the value may be negative. Even numbered field lengths do not allow a negative value. Negative values are displayed with a minus sign (-) preceding the number. **D** long date field is 10 characters in size. A space or zero may precede a single digit day or month in

the format MM/DD/YYYY. **S** short date field is 8 characters in size in the format of MM/DD/YY. **T** time field is 8 characters long in the format of HH:MM:SS. This is any legal alphanumeric field in the format of *fieldname[ss]* or alphanumeric constant. Required.

**size** - This is the number of characters in the field. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**dec chrs** - If a N type field has been specified then this is the number of decimal characters that will be allowed. Must be 0,2,4,6, or 8. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant.

**file num** - If this field is to be redefined so that it will be attached to a file buffer then this would be the appropriate file number. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant.

**offset** - If a file number has been specified above then this is the starting location within the file buffer. The first character position in the buffer is 1. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

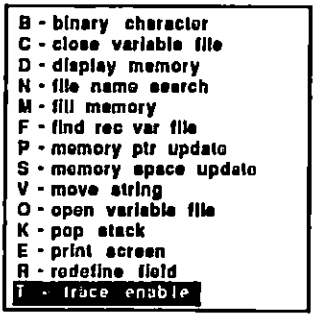
**key num** - If this field is to be treated like a key then this is the appropriate key number. Effective only for those fields where a file number is specified. This is any legal numeric field in the format of *fieldname[ss]* or numeric constant. Required.

**TAS EDITOR EXAMPLE:**

```
redefine field: field name: NEW.FIELD type:
NEW.TYPE size: NEW.SIZE dec chrs: NEW.DEC file
num: FILE offset: FILE.OFFSET key num: KEYN

redefine field: field name: NEW.FIELD type: 'A' size:
10
```

# trace enable




This command will place a break point at the current line location within the program. It will also cause the compiler to produce a file with the extension .SYM that will contain a listing of each of the line labels and the numbers for each of the fields. For more information on the **trace utility** please refer to Chapter VII in this manual.

**PARTS OF THE COMMAND:**

**none**

**TAS EDITOR EXAMPLE:**

trace enable

# II. Accessing Files Using Variable File Numbers

TAS-Plus now offers the programmer another significant power feature. This is the capability of accessing files using just the file number. This allows you to manipulate data files without knowing ahead of time what the file name is.

Most of the commands that directly affect a data file have been enhanced from the normal TAS-Plus version. In some cases new commands have been created. You can now use a field instead of a constant to specify the file number and use it where you would normally put the file name. The only thing that changes is that you will precede the field name with an at sign (@). For example, let's say you want to use a file number in the **save record** command. Normally, you would put the file name in say, GLDATA, when the editor prompts you for it. In this case though, you want to use the field FILE.NUM. Instead of entering the file name you would enter @FILE.NUM. Everything else will remain the same. The commands which will allow you to use this method are:

**delete record**
**equals active records**
**equals record num**
**equals total records**
**save record**

The new commands that have been created and are listed in further detail in this manual are:

**close variable file**
**find rec var file**
**open variable file**
**redefine field**

The normal procedure would be to open a file using the **open variable file** command and then accessing records using the **equals record num** or **find rec var file** command.

You may open as many files using this method as you do using the normal open command. However, if you want to open and then subsequently close these files in preparation for opening others you must close the last opened file first. This will allow TAS-Plus to reclaim the buffer space used by that file and will allow it to be reused by the next open.

The **redefine field** command will allow you to define portions of the file as fields so that they may be accessed just like they would in a normal TAS-Plus program. Even key fields may be defined. For examples of this technique you should look at the programs TASBRWSE and TASADREC. Both use the variable file procedures.

# III. Memory Storage Areas

## GENERAL EXPLANATION

Another new feature in TAS-Plus is the ability to access large areas of memory. We have setup 4 contiguous memory blocks. Each block contains 24k of memory. These blocks may be accessed using the following commands:

**display memory**
**equals portion of**
**fill memory**
**memory ptr update**
**memory space update**
**move string**
**put portion of**

By having access to these memory locations you are able to keep much more than the 255 field limit. This also allows you to exceed the limit of 255 array elements since each element could be just another group of data in one of the memory areas.

Data can stretch across memory areas. For example, if you want to keep a single block of data that will be 40k long you could start it in memory area 2 and it would automatically overlap into area 3. You would still access it as though it were in area 2 however. This works very well but you obviously need to remember not to put any data into the lower regions of area 3.

Something else you want to remember is that the **equals portion of** and **put portion of** commands can transfer exact images of the field not the ASCII representation. This means that when moving numeric, date, or time fields the actual data will be less then the defined size. For example, a number defined to be 8 characters long would actually only be 4 characters when stored. To determine the calculated size (also referred to as csize) of a number you divide the defined size by 2. If there is a remainder add 1 to the result. If the defined size is 5 the csize will be 3. A date, regardless of type, will always have a csize of 4 and time fields will always be 3. You can always move more but it just wastes space and when you move it back you will not transfer the correct value.

## COMMON MEMORY

Another very useful feature of this memory is to utilize it as a command storage area. Once TAS-Plus is loaded these areas will not be changed until it is loaded again. This includes even executing a different .EXE or .COM program from within TAS-Plus or a DOS command.

This means that you can safely store information in one of the memory areas and know for a fact that it will still be there when you reach your next program. A method we use is to put a binary 255 (0FFh) in the first position of the first memory area. We accomplish this by using the **equals** command and setting a one character alphanumeric field equal to %255. (**NOTE:** You may precede a number with a percent sign in an expression and the binary value of that number will be placed into the receiving field. The receiving field should be a 1 character alphanumeric.) We then place that field into the first position of the memory area we're using, generally memory area 1, using the **put portion of** command (don't forget to use a dummy field as the receiving field name). We then follow the character with whatever information we want to transfer. This is the method we use for making TASBRWSE act as the Maintain Locator File program. You just don't want to forget to clear the marker immediately after you use the common information. This will keep you from thinking that there is still more information in the common area after it has been used.

### Changes to equals portion of and put portion of commands

These two commands will allow a memory area location to be entered. This feature is used when you want to move data into or out of one of the memory areas. The rest of the command would be the same the only difference being the addition of the locatic number. You should also put a dummy field name as the receiving field in the case of **put portion of** or the field to be parsed in the case of **equals portion of**. We define a field called DUMMY for just that purpose.

# IV. Conversion of TAS-Plus Editor Commands to ASCII Text Editor Type Commands

If you've looked at all at the .EDT or .SRC files provided with this copy of TAS-Plus you will notice that they are much different. When TAS was first released the only method of creating a program was to use an ASCII text editor, for example Wordstar. With the release of TAS-Plus you now have a full syntax free editor that you may use instead. However, many people would rather use a standard text editor and so we have provided you with the means to use either method.

The only major difference between the two is that in a non TAS-Plus edited program you will not be able to easily use color attributes on screen formats. However, you can send color attributes to the screen with the **print message** command. The following are the command titles in TAS-Plus and what the command would look like in the text editor method. We have used the same option titles as in the standard TAS-Plus manual and you should exchange those for your entries.

For many examples of the use of these codes please refer to the programs included. We use the convention of naming the files edited with the TAS-Plus editor with the extension .EDT and the text edited files with .SRC. However, since the compiler will accept any extension, including none at all, you may use any convention you like.

The first line will show the TAS-Plus command name and the subsequent line(s) will be the ASCII editor format.

NOTE: All options in a command line are separated by commas. If you have no more entries to make you may put the closing parenthesis after the last option. You do not have to put all of the commas in. However, if you want to use an option later in the command line then you do have to put the appropriate intervening commas in. For example: enter(TEST) is a legal command, however, if you wanted to force it to upper case then the following is what you would do: enter(TEST,,,,Y).

The maximum length of any command line is 255 characters and only one command may be on a line except in the case of the THEN option in the IF command where the next command to execute would follow on the same line as the IF command. You may also have a line label and a command on the same line. The parts of the command listed below in the non-TAS editor notation have the same definitions as in the TAS source code editor. For further information please refer to the TAS-Plus reference sections.

---

bell

bell

---

binary character

bin(alpha field,numeric field,what to do)

---

clear array

clrmem(mem field,number of elements)

---

clear buffer

clr(filename,what to clear)

---

clear line

clrlne(column,row)

---

clear screen

clrscr

---

close non-tas file

clso(delete after close)

---

close tas file

close(filename)

---

close variable file

closev(file num)

---

company code

co(fieldname,get or set)

---

compile program

comprg(filename,print results to)

---

cursor on/off

cursor(YN)

---

date

date(field name,get or set)

---

define field

define(field name,type, size/filename,decimal chrs, array elements)

---

delete file

delf(file name)

---

delete record

del(file name,confirm,delete all records,goto label)

---

display array

dispf(field name,offset, column,row,width,number of lines)

---

display memory

dispm(mem loc,start loc,element length,offset,column,row, width,number of lines)

---

else

else

---

enter

enter(field name,mask string,help label,up arrow label,upcase, auto cr,password type,enter at column number, enter at row number)

---

equals

receiving field = (expression)

---

equals active records

receiving field =nrc(filename)

---

equals day

receiving field =day(date field)

---

equals location

receiving field =loc(chrs to search for,field to search within, starting character number)

---

equals month

receiving field =xmt(date field)

---

equals portion of

receiving field =mid(use field,starting character number,number of chrs to move,memory area number)

---

equals record num

receiving field =rcn(file name,get or set)

---

equals size

receiving field =siz(which type,field being checked)

---

equals total records

receiving field =trc(file name)

---

file name search

fnsrch(file name,first/next)

---

fill field

fill(field to be filled,leading or trailing,character to be used)

---

fill memory

fillmem(mem loc,start,num chrs,fill chr)

---

find record

find(find type,field name/file name,related filename,on error label)

---

find rec var file

findv(file num,key num,search val,search type,error label)

---

for/next

for(counter,start,stop,step)

---

get character (get/set chr & atr)

scrchr(column value, row value,character field,atribute field,get or set chr and atr)

---

get field from buffer

receiving field =gfd(delimited field number)

---

gosub

gosub label

---

goto

goto label

---

if

if compared field <,<=,=,=>,<> (expression) action

---

if carriage return

ifcr(what to do,line label)

---

if duplicate

ifdup(field name,what to do,line label)

---

if rec not found

ifna(file name,what to do,line label)

---

initialize file

inifle(file name)

---

justify field

just(field name,justify left/right/center)

---

line label

in ASCII editor version you just put a colon (:) after the label name

---

memory ptr update

memptr(pointer size,start loc,mem loc,amt to add/sub,occurs every # of chrs,how many ptrs,what to do)

---

memory space update

memspc(mem loc,number of chrs to add/sub,start loc,stop loc, what to do)

---

menu

menu(lower left column value,lower left row value,height,width, menu array field,choice field,maximum number of choices, maximum width of choice,where to go if ESC is pressed,where to gosub if HELP is pressed,menu title)

---

mount

mount(format name,which type,where to print,file name)

---

move string

xfer(frm s/l#,frm offset,to s/l#,to offset,num of chrs)

---

on value

on(on field,what to do,label1,label2,...,labelx,error label)

---

open non-tas file

opno(type of file,file name,buffer start,record size)

---

open tas file

open(file name,company code,lock type,error label)

---

open variable file

openv(file name,file num)

---

pop return off stack

pops

---

print chr$

pchr(character string number)

---

print blank lines

pblnk(number of lines)

---

print box

pbox(column, row,height,width,clear box)

---

print message

pmsg(message,column value,row value,wait for chr entry,do not do a carriage return,force output where,entry field)

---

print on

pon(print on which device,file name)

---

print outside window

force(Y/N)

---

print report line

pfmt(report format line number, column value, row value, wait for chr entry,do not do a carriage return,force output where)

---

print screen

prtsc

---

print set

pset(max width,max lines,max print lines)

---

print vertical tab

pvert(line number)

---

put field in buffer

pfld(field to be added to record)

---

put portion of

mid(receiving field,starting character,number of characters, field to be inserted,memory area number)

---

read non-tas rec

rdrec(on error goto label)

---

redefine field

redef(field name,field type,size,dec chrs,file num,offset,key num)

---

redisplay screen

redsp(buffer number)

---

reenter field

reent

---

remark

;

---

rename file

renf(current name,new name)

---

return

ret

---

reindex file

rkyfle(file name)

---

run dos command

dos(command line)

---

run non-tas program

exec(program name,command line)

---

run tas program

run(program name)

---

save record

save(file name,confirm,clear buffer,goto label)

---

save screen

saves(buffer number)

**NOTE:** A facility available in the Developer's version allows you to save the screen to a memory area. If you add 20 to the memory area number then the program will save the entire screen, characters and attributes. If you add 40 to the memory area number the program will save just the characters.

---

screen lock/unlock

scrn(what to do)

---

scroll

scroll(column,row,height,width,numof lines/columns,which way)

---

set color

color(normal color,highlighted color,reverse color)

---

set file active

setact(schema name,file name)

---

set video highlight

bkg

---

set video normal

frg

---

set video reverse

rev

---

structure start

{

---

structure stop

}

---

time

time(field name,get or set)

---

top of form

tof

---

trace enable

break

---

trap

trap(type of trap,what to do,label)

---

trim field

trim(field name,leading or trailing)

---

upcase field

upcase(field name,all or quotes)

---

while

while compared field <,<=,=,=>,<> (expression)

---

window

window(column,row,height,width)

---

wrap print

wrap(print continued line at)

---

write non-tas rec

wtrec(on error goto label)

---

# V. Creating Screen and Report Formats for TAS-Plus Using an ASCII Text Editor

If you are going to use an ASCII text editor to create your programs there is a slightly different method for creating screens or report formats for your programs.

You can create screens and report formats using the source code editor and then move them into your programs. You must, however, do this for all the screens and report formats in a program if you choose to do this for any. To do this you would first create the screen using the source code editor. Then, using your ASCII text editor load the program you created. The beginning of the screen will start with a backslash (\S or \R). You want to include everything from that point inclusive until you reach the 3rd backslash. This would include the first part which is the screen and the second part which is the field information. Make sure that you include the last (3rd) backslash. You could then insert the entire screen in your program at any location. Don't forget the **mount** command and don't change anything in the screen. If your program doesn't compile or the compile hangs you probably didn't include all the backslashes. NOTE: You will notice in screen formats that the first three characters on the first line will be \SS or \ST. Do not change it at all. The compiler needs that information to determine what type of screen it is compiling. The title of the screen starts with the next character. For example, a screen with the name TEST created in the source code editor using the color attributes would have a first line as follows: \SSTEST.

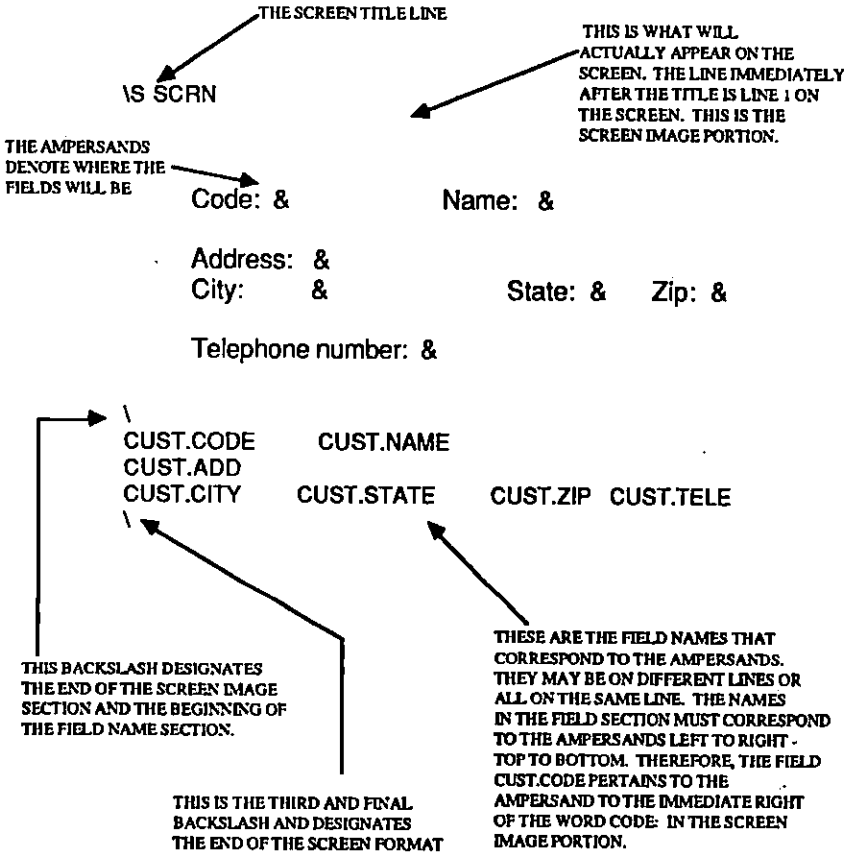
If you wish to create screens or report formats with your ASCII text editor then you would do the following. A screen or report format consists of 3 different parts. The first is the title line. It starts with a backslash with the next character an S if it is a screen format or an R if it is a report format. The difference between screen formats and report formats is that screen formats print as a group on the crt screen. The format is taken as a whole instead of individual lines. The line immediately after the title line is considered line 1. In a report format each line is considered individually. Once again,

the line immediately after the title line is considered line 1 but you might use line 5 before you do line 1. This is determined by the number you use in the **print report format** command. You may print any line in any order. The maximum size for a screen is 80 columns wide by 24 lines long. Anything outside those dimensions is ignored. A report format may be 128 columns wide by 66 lines long. You designate where fields will appear on the screen or report by placing an ampersand (&) in the appropriate location.

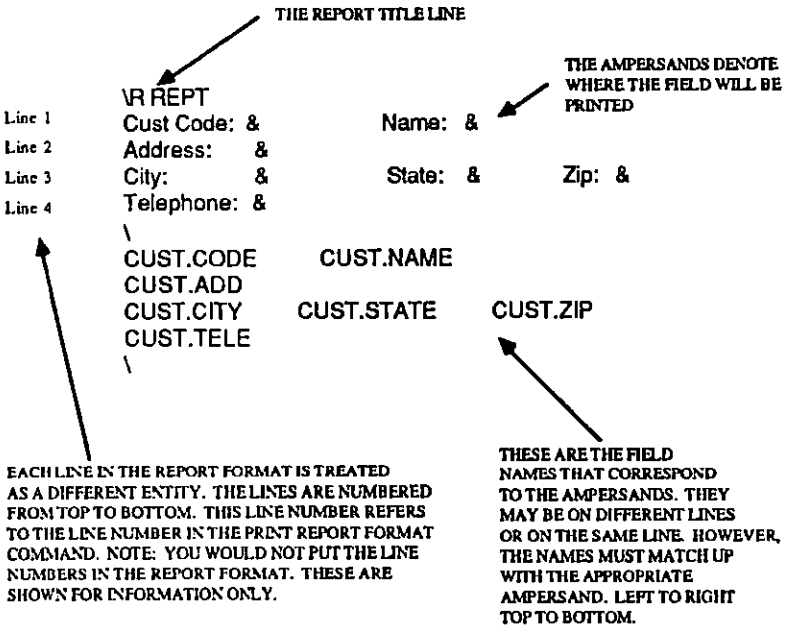
The end of the format is designated by a second backslash in the first column with nothing after it. The third and final part of the format is the field information section. For each ampersand in the screen/report section you must have a corresponding field name. You match the field names to the ampersands in a left to right top to bottom order. Field names may be one to a line or all on the same line separated by spaces or anything in between. We generally put the fields that appear on the same line in the format, on the same line in the field information section. This helps to make sure that all fields are accounted for.

The final part of the field section is the third backslash. Once again this would be in the first character position and would have nothing after it. You may put another format immediately following or any other code. Or the format may be placed at the end of the file. Make sure that there is a carriage return after the final backslash.

The following is an example of a screen format and a report format.




# Schematic of a Screen Format




# Schematic of a Report Format

# VI. Cross Reference Listing of TAS-Plus Command Numbers and Titles

If you have looked at a .EDT source file you will notice that all lines start with a 3 character number. This number is the internal representation of the command name. The following is a list by number of the command names and following that a list by command name of the numbers.

| Num | Command Title |
|---|---|
| 000 | line label |
| 001 | remark |
| 002 | enter |
| 003 | open tas file |
| 004 | open non-tas file |
| 005 | equals |
| 006 | equals location |
| 007 | equals portion of |
| 008 | equals record num |
| 009 | equals day |
| 010 | equals total records |
| 011 | equals active records |
| 012 | get field from buffer |
| 013 | equals month |
| 014 | trim field |
| 015 | fill field |
| 016 | justify field |
| 017 | upcase field |
| 018 | if |
| 019 | if duplicate |
| 020 | if rec not found |
| 021 | goto |
| 022 | gosub |
| 023 | on value |
| 024 | save record |
| 025 | delete record |
| 026 | clear buffer |
| 027 | find record |

| | |
|---|---|
| 028 | read non-tas rec |
| 029 | write non-tas rec |
| 030 | put field in buffer |
| 031 | print message |
| 032 | print report line |
| 033 | print set |
| 034 | print vertical tab |
| 035 | print blank lines |
| 036 | print on |
| 037 | print chr$ |
| 038 | print box |
| 039 | trap |
| 040 | run tas program |
| 041 | screen lock/unlock |
| 042 | for/next |
| 043 | while |
| 044 | not used at present time |
| 045 | mount |
| 046 | define field |
| 047 | structure start ({) |
| 048 | structure stop (}) |
| 049 | else |
| 050 | return |
| 051 | close tas file |
| 052 | top of form |
| 053 | close non-tas file |
| 054 | clear screen |
| 055 | set video highlight |
| 056 | set video normal |
| 057 | clear line |
| 058 | bell |
| 059 | not used at present time |
| 060 | set file active |
| 061 | reenter field |
| 062 | clear array |
| 063 | time |
| 064 | date |
| 065 | set video reverse |
| 066 | set color |
| 067 | save screen |
| 068 | redisplay screen |
| 069 | company code |

| | |
|---|---|
| 070 | run non-tas program |
| 071 | window |
| 072 | menu |
| 073 | cursor on/off |
| 074 | put portion of |
| 075 | if carriage return |
| 076 | get/set chr & atr |
| 077 | compile program |
| 078 | initialize file |
| 079 | reindex file |
| 080 | scroll |
| 081 | rename file |
| 082 | delete file |
| 083 | memory ptr update |
| 084 | memory space update |
| 085 | trace enabled |
| 086 | pop return off stack |
| 087 | not used at present time |
| 088 | wrap print |
| 089 | print outside window |
| 090 | equals size |
| 091 | run dos command |
| 092 | display memory |
| 093 | display array |
| 094 | binary chr |
| 095 | open variable file |
| 096 | find record variable file |
| 097 | close variable file |
| 098 | fill memory |
| 099 | redefine field |
| 100 | print screen |
| 101 | move string |
| 102 | file name search |

| Command Title | Num |
|---|---|
| bell | 058 |
| binary chr | 094 |
| clear array | 062 |
| clear buffer | 026 |
| clear line | 057 |
| clear screen | 054 |
| close non-tas file | 053 |
| close tas file | 051 |
| close variable file | 097 |
| company code | 069 |
| compile program | 077 |
| cursor on/off | 073 |
| date | 064 |
| define field | 046 |
| delete file | 082 |
| delete record | 025 |
| display array | 093 |
| display memory | 092 |
| else | 049 |
| enter | 002 |
| equals | 005 |
| equals active records | 011 |
| equals day | 009 |
| equals location | 006 |
| equals month | 013 |
| equals portion of | 007 |
| equals record num | 008 |
| equals size | 090 |
| equals total records | 010 |
| file name search | 102 |
| fill field | 015 |
| fill memory | 098 |
| find record | 027 |
| find record variable file | 096 |
| for/next | 042 |
| get/set chr & atr | 076 |
| get field from buffer | 012 |
| gosub | 022 |
| goto | 021 |
| if | 018 |

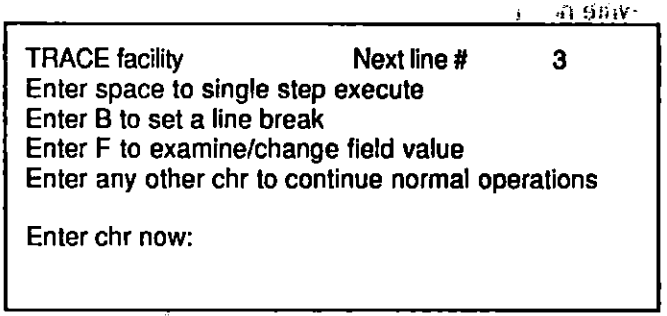
| | |
|---|---|
| if carriage return | 075 |
| if duplicate | 019 |
| if rec not found | 020 |
| initialize file | 078 |
| justify field | 016 |
| line label | 000 |
| memory ptr update | 083 |
| memory space update | 084 |
| menu | 072 |
| mount | 045 |
| move string | 101 |
| on value | 023 |
| open non-tas file | 004 |
| open tas file | 003 |
| open variable file | 095 |
| pop return off stack | 086 |
| print chr$ | 037 |
| print blank lines | 035 |
| print box | 038 |
| print message | 031 |
| print on | 036 |
| print outside window | 089 |
| print report line | 032 |
| print screen | 100 |
| print set | 033 |
| print vertical tab | 034 |
| put field in buffer | 030 |
| put portion of | 074 |
| read non-tas rec | 028 |
| redefine field | 099 |
| redisplay screen | 068 |
| reenter field | 061 |
| remark | 001 |
| rename file | 081 |
| return | 050 |
| reindex file | 079 |
| run dos command | 091 |
| run non-tas program | 070 |
| run tas program | 040 |
| save record | 024 |
| save screen | 067 |
| screen lock/unlock | 041 |

| | |
|---|---|
| scroll | 080 |
| set color | 066 |
| set file active | 060 |
| set video highlight | 055 |
| set video normal | 056 |
| set video reverse | 065 |
| structure start ({) | 047 |
| structure stop (}) | 048 |
| time | 063 |
| top of form | 052 |
| trace enable | 085 |
| trap | 039 |
| trim field | 014 |
| upcase field | 017 |
| while | 043 |
| window | 071 |
| wrap print | 088 |
| write non-tas rec | 029 |

# VII. Trace Utility

A trace utility has been included in TAS-Plus. This feature will allow you to put break points into your source code and single step run from there or continue until a line is reached or another break point. You will also be able to see and change field values.

The first step is to place a **trace enable** (break point) command into your source code at the appropriate location. This has two functions. The first is to tell the compiler to create a file with the same name as your source file but with the extension .SYM. This file will contain a listing of each of the fields and line labels specified in the program and their appropriate numbers. This will be necessary when using the trace facility during run time. The compiler will also put a break point at that location. When that line is executed the program will open a window like the following:




Initial Trace Screen

The next line number will correspond to the run time line number. This is the second row of numbers printed when you printout a source code listing using the source code editor. You will also have the choice of entering a B, F or just pressing the space bar. If you enter any other character the program will continue executing until the next trace enable command is encountered.

To set a break point further in the program you would enter a **B**. The program will then prompt you for a line number. Any line may be used. Once the break point is encountered it will be automatically cleared unless you reset it.

If you want to examine or change a field value then enter F. The program will then ask you for the field number. If it is an array field you will also be able to enter the array number. The program will then display the current field value in the lower left corner and you will be able to change it if desired. After accepting or changing the field the program will return to the initial choices.

The third option is to single step through the program. This is useful for trying to trace the program's path. This is accomplished by pressing the space bar only. The trace window will be displayed at each line and you will be able to execute any of the above choices or single step to the next line.

# VIII. Multiuser Operations

If you have purchased the multiuser version of TAS-Plus then the installation program will have an additional option. You will be able to designate whether the program is to be run in multiuser mode or not. If it is, then answer 'Y' to the appropriate question.

TAS-Plus uses the MS-DOS/Netbios 3.1 version of file and record locking. Most LAN networking systems and multiuser operating systems are being released compatible to this standard.

A file is opened as shared automatically in a TAS-Plus program. If you want to enforce record and or file locking then you must specify the appropriate response when opening the file. Please refer to the reference section of the main TAS-Plus manual. The **open tas file** command. If you open non-TAS files they are automatically opened as exclusive files for both reading and writing.

If two users try to access the same record at the same time TAS-Plus will lock out the second user until the first has released the record. This may be done by writing the record back and clearing the buffer or just clearing the buffer. You may also clear the record number under program control for the same results. Also, if you read another record in the same file then a record already locked will be released first.

You may override the automatic wait by using the **trap** command, **L** option. If this trap is set then the program will transfer control to the appropriate line if a record is found to be locked by another user. You can then have a message be displayed and try to lock the record again, or go to a different location for further entry. You should be careful that you release all other locks that may have already been successfully placed if you are accessing multiple files.